Midterm Test #1 *(Sunday 26-04-2015)* **Time Allowed 90 Minutes**

Answer all Questions. **TOTAL MARKS 20**

الإجابة النموذجية

Q1- (5 pts)

a) Explain how C compiler deals with overflow in addition and subtraction operations. ***(1.5 pts)***
C Compiler ignore overflow on MIPS processor by using addu, addiu & subu instructions.

b) How to improve the speed of a 32-bit ripple adder to become 24-gate delay instead of 64-gate delay. *(Assume that the gate delay in the carry path of each full adder is 2-gate delay)*. ***(1.5 pts)***
This can be done by connecting 8 x 4-bit CLA adders in ripple; each 4-bit CLA has 3-gate delay.

c) Explain how to implement **`slt`** instruction in a 32-bit ALU. ***(2 pts)***
To implement slt instruction in a 32-bit ALU, the adder inside the ALU will perform subtraction operation, the sign bit will xor-ed with the overflow, the output of the xor gate will be the result of the least significant bit R[0] , the other bits R[1] → R[31] tided to ground (Equal zero).

Q2- (7 pts)

Write MIPS assembly programs that perform the following:

a) Invert the contents of **`$s2`** using only one instruction. ***(1 pts)***
NOR instruction is used :

```
nor $s2,$s2,$0   (nor $s2,$s2,$s2)
```

b) POP registers **`$s0`** and **`$s1`** respectively from the stack memory. ***(1.5 pts)***

```
lw $s0,0($sp)
lw $s1,4($sp)
addi $sp,$sp,8
```

c) Execute **`(bne $s3,$s4,L1)`** if the branch target is too far to encode with 16-bit offset. ***(2 pts)***
If the branch target address is too far to encode with 16-bit offset we can use jump instruction and beq instruction instead of bne as following:

```
    beq $s3,$s4,L2
    j L1 # L1 may be forward or backward
L2:
```

d) Translate the following C code to MIPS assembly program using only five instructions. *(Assume A and B are 32-bit signed integer, and they are assigned to $s5 and $s6 respectively).* ***(2.5 pts)***

```
if(A >= B)
 A++;
else
 B--;
```

```
      slt $t0,$s5,$s6   # $t0 = 1 if A<B
      bne $t0,$0,else   # Branch if A<B
      addi $s5,$s5,1    # A++
      j Exit            # Jump to exit to avoid executing B--
else: addi $s6,$s6,-1   # B--
Exit:
```

Q3- (4pts)

Given a program running on a computer M1 with 2GHz clock frequency, and 10^9 instructions divided into three classes as follows:

	Arith/Logic Class	Load/Store Class	Branch/Jump Class
IC	60%	30%	10%
CPI	4	20	3

We are trying to improve the execution time without changing in the cycle time; this can be done in hardware by decreasing of 75% in the clock cycles per instruction of load/store class.
Calculate:

a. CPU cycles, and CPU time before and after improvement. ***(3pts)***

Before Improvement:

$$CPU\ cycles = \sum_{i=1}^{3} CPI_i \times IC_i = (4 \times 0.6 + 20 \times 0.3 + 3 \times 0.1) \times 10^9 = 8.7 \times 10^9\ cycles$$

$$CPU\ Time = \frac{CPU\ Cycles}{Clock\ Rate} = \frac{8.7 \times 10^9}{2 \times 10^9} = 4.35\ sec$$

After Improvement:

The CPI of Load/Store is decreased 75% → CPI = 5 Clock cycles per instruction

$$CPU\ cycles = \sum_{i=1}^{3} CPI_i \times IC_i = (4 \times 0.6 + 5 \times 0.3 + 3 \times 0.1) \times 10^9 = 4.2 \times 10^9\ cycles$$

$$CPU\ Time = \frac{CPU\ Cycles}{Clock\ Rate} = \frac{4.2 \times 10^9}{2 \times 10^9} = 2.1\ sec$$

b. *The speed up.* ***(1pt)***

$$Speed\ up = \frac{CPU\ time\ before}{CPU\ time\ with\ improvement} = \frac{4.35}{2.1} = 2.071$$

Q4- (4pts)

Complete the steps of the following multiplication as shown in the table below.

Iterations	1st Version unsigned multiplier			
	Steps	MR_{reg}	MC_{reg}	PR_{reg}
	Initial Values			000000
1				000100
2				001100
3				001100
Stop	MR = _____, MC = _____, PR = ____			

Solution:
Three iterations → Multiplier size is 3x3 bit (MR and MC are 3bits, and PR is 6 bit)

Iteration 1 : Product register = 000100 → PR_{reg} = PR_{reg} + MC_{reg} → 000100 = 000000 + MC_{reg} → MC_{reg} = 000100, and MC = 4.
ALSO because PR_{reg} is changed this means that the bit MR[0] = 1.
Then we do the shift operation for MC_{reg} and MR_{reg}.
In this case MC_{reg} = 001000
But MR_{reg} still unknown.

Iterations	*1st Version unsigned multiplier*			
	Steps	*MR_{reg}*	*MC_{reg}*	*PR_{reg}*
	Initial Values	*???*	*??????*	*000000*
1	*PR=PR+MC* *SH_L MC, SH_R MR*	*??1* *???*	*000100* *001000*	*000100*
2				*001100*
3				*001100*
Stop	*MR = _____, MC = _____, PR = ____*			

Iteration 2: Product register = 001100 → PR_{reg} = PR_{reg} + MC_{reg} → MR[0] in this iteration = 1.

Iterations	*1st Version unsigned multiplier*			
	Steps	*MR_{reg}*	*MC_{reg}*	*PR_{reg}*
	Initial Values	*???*	*??????*	*000000*
1	*PR=PR+MC* *SH_L MC, SH_R MR*	*??1* *0??*	*000100* *001000*	*000100*
2	*PR=PR+MC* *SH_L MC, SH_R MR*	*?11* *00?*	*010000*	*001100*
3	*SH_L MC, SH_R MR*	*000*	*100000*	*001100*
Stop	*MR = _____, MC = _____, PR = ____*			

Iteration 3: Product register = 001100 → PR_{reg} = PR_{reg} → MR[0] in this iteration = 0.

Thus: MR = 3, MC =4 and PR =12

Iterations	*1st Version unsigned multiplier*			
	Steps	*MR_{reg}*	*MC_{reg}*	*PR_{reg}*
	Initial Values	*???*	*??????*	*000000*
1	*PR=PR+MC* *SH_L MC, SH_R MR*	*??1* *0??*	*000100* *001000*	*000100*
2	*PR=PR+MC* *SH_L MC, SH_R MR*	*?11* *00?*	*010000*	*001100*
3	*SH_L MC, SH_R MR*	*000*	*100000*	*001100*
Stop	*MR = 3, MC = 4, PR = 12*			